UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering

EEC 172 **Spring 2021**

LAB 4: Introduction to AWS and RESTful API

Lab Due: May 16, 2021

Objective:
In this lab, you will explore the RESTful API and use it to connect with Amazon Web Services (AWS). First, you will create a 'thing' in AWS and use the HTTP GET command to retrieve status information about your thing. You will then use your code from Lab 3 to compose messages with an IR remote, which you will send to AWS using the HTTP POST command. You will use a rule in AWS to send the text to your cellphone using Amazon's Simple Notification Service (SNS). Much of the code you will need for accessing AWS will be provided to you. You will need to understand it at a high-level and modify it for your specific AWS account.

Please view the video AWS IoT Getting Started to understand the basics of AWS IoT.

New Equipment Needed

- An Amazon AWS account (You can get a 1-year free trial account) See https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-free-tier.html for information on using the AWS free tier.

Development Tool Installation Procedure
To do the lab on your own computer, you will need to install the following programs:

1. **OpenSSL** (https://www.openssl.org/)
   OpenSSL will be needed to convert the .pem formatted certificate and keys to the .der format that is required. If you have other means of doing this, that is fine as well.

Background on the RESTful API:

The RESTful API, based on the Representational state transfer (REST) architecture, is a web-based communication protocol that is widely used for many services. The RESTful API is typically implemented using HTTP. Although not as optimized for low power and low-cost applications as another IoT protocol, MQTT, RESTful API is widely used due to its simplicity and genericity.

The RESTful API defines the format of the messages that are sent to a service to interact with that service. The function of the RESTful APIs will vary depending on the service with which you are interacting. Some of the commonly used commands are listed below.

Common RESTful API Commands:

- **GET** - Often used to retrieve data or information from a service. May also just be used as a simple trigger. For AWS IoT, GET can be used to retrieve a device's parameters from its shadow.
- **POST** - Can be used to update or push data to the cloud or service. For AWS IoT, POST could be used to update a device's parameters in the device shadow.
- **DELETE** - Used to manage and remove data structures in the cloud. For AWS IoT, DELETE can be used to delete a device shadow.

Other RESTful API commands may exist, depending on the service, and are based on HTTP commands: *HEAD, PUT, TRACE, OPTIONS, CONNECT, PATCH.*

For more information on the RESTful API please see: Beginners guide to creating a REST API: http://www.andrewhavens.com/posts/20/beginners-guide-to-creating-a-rest-api

Background on AWS IoT:

Amazon Web Services (AWS) is a collection of various web-services for your cloud-based needs and removes the need of individual businesses to develop the hardware infrastructure for these initiatives. Recently, the Internet of Things (IoT) service was released, allowing connection of many hardware products.

To represent these *real-world* devices in the cloud and maintain a persistent/coherent state with intermittent network connections, AWS IoT has a concept of a device *shadow*. This shadow represents the *real-world* device's state in the cloud. As the device changes states, it pushes its changes to the shadow device on AWS via MQTT or a RESTful API. If the device loses network connection, and then comes back online, the device can pull the last-known state from the shadow to update itself. Updates to this shadow can trigger actions in other services as well using *Rules*. In this lab, a rule is used to send text messages to your cell phone.

Part I: Connecting to AWS (securely) and updating your Shadow

Part I.A: Setting up an Amazon AWS account

For this lab, you will need to setup an Amazon AWS account and get a basic understanding of AWS. One person per team will need to create an AWS account that can be shared for completing this lab.

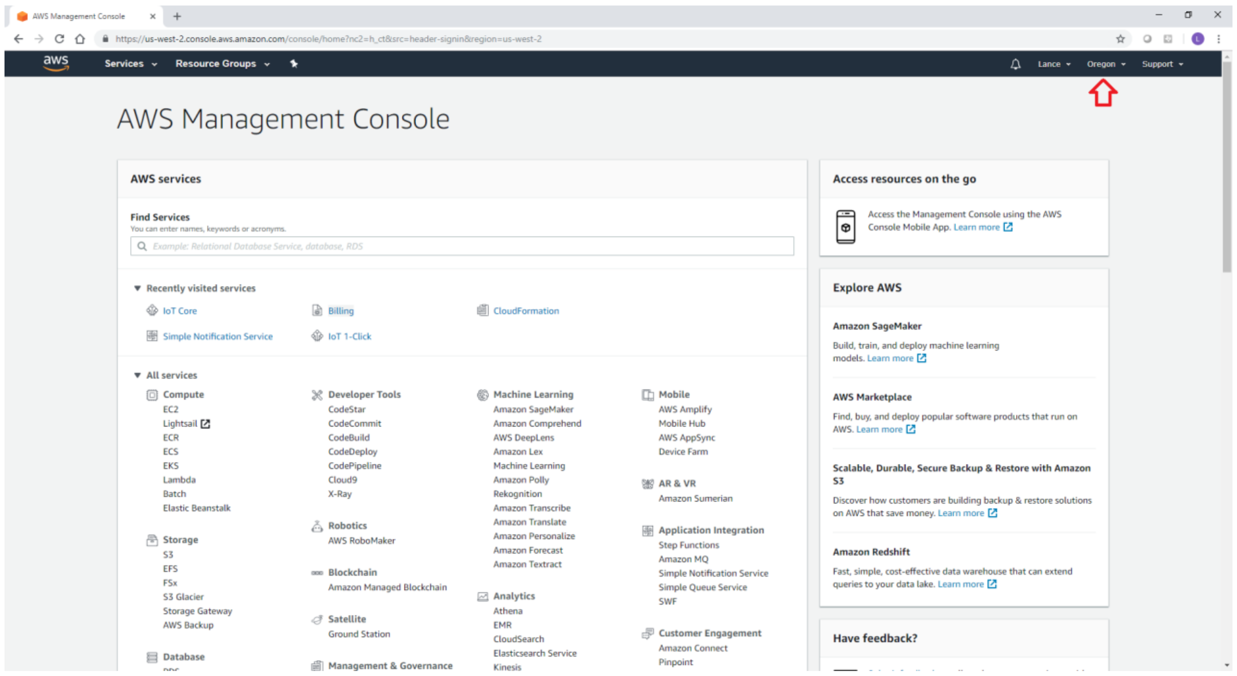


Figure 1: Amazon AWS Console View

After setting up an account, log into the console. You should end up seeing a page similar to Figure 1. After you are logged in, expand the services and click on the **AWS IoT Core** link. If it does not appear, make sure you have selected an Amazon AWS region at which this service is active. We recommend that you use the **U.S. West (Oregon)** AWS server for this lab.

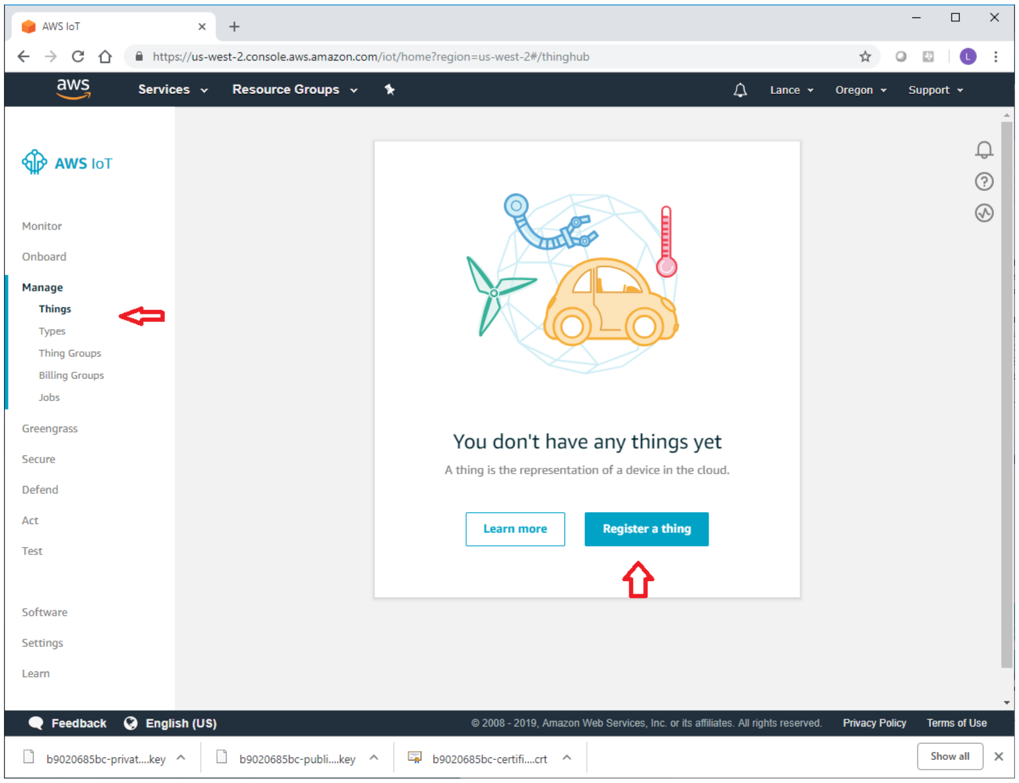
In the AWS IoT Core service, the controls on the left will list the actions you can do such as Monitor, Onboard, Manage, Secure and Act. In the bottom left corner of the screen, there is a *Learn* button and a link to Amazon's IoT *interactive* tutorial. Click on it, and go through the tutorial for a *high-level explanation* about how AWS IoT uses Rules to do cool things.

Part I.B: Setting Up Your First Device Thing/Shadow with the AWS IoT Console

For the next part of the lab, you will need to generate the access keys to allow your CC3200 to connect to your Amazon AWS account securely. For a *real-world* device to connect to Amazon AWS you will need to do the following:

1. **Create a Device Thing/Shadow** This will generate a server-side representation of your *real-world* device (your thing).
2. **Generate the required certificates and keys** for establishing an encrypted connection
3. **Attach the certificate and keys** to the Device Thing/Shadow
4. **Give the Device Thing/Shadow permission** to use the IoT functions of AWS

Fortunately, the above can be accomplished using the Amazon Console interface. Click on **Manage,** then **Things** and then **Register a thing** as shown in Figure 2.



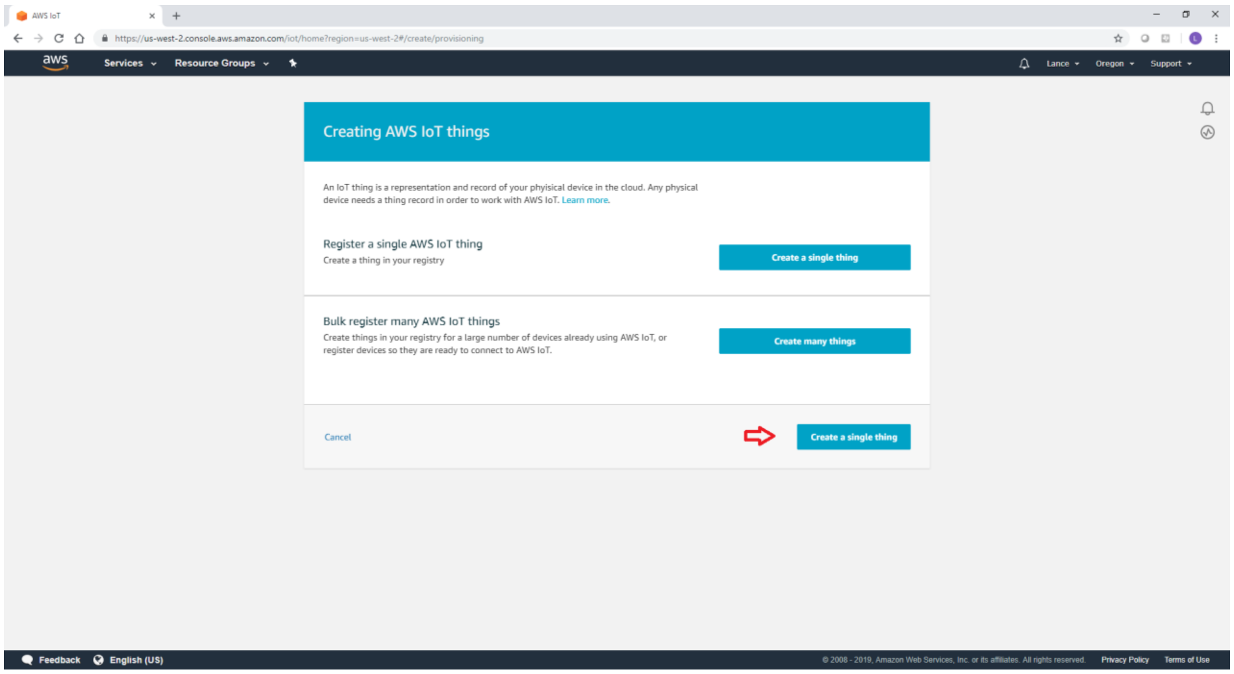
You don't have any things yet

A thing is the representation of a device in the cloud.

© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 2: Creating Your First Thing

You should be presented with a display similar to Figure 3. Click on the button at the bottom to **Create a single thing**.


AWS IoT
https://us-west-2.console.aws.amazon.com/iot/home?region=us-west-2#/create/provisioning
Services
Resource Groups
Lance
Oregon
Support
Create a single thing
Create many things
Cancel
Create a single thing
Feedback
English (US)
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Creating AWS IoT things

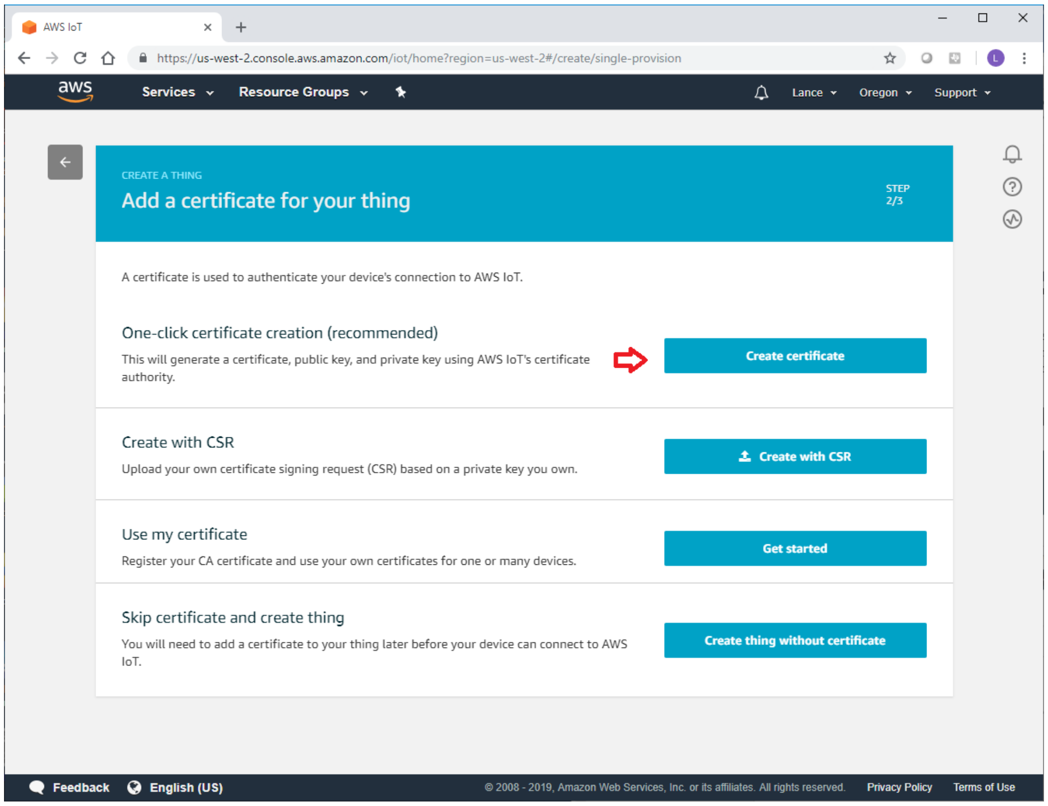
An IoT thing is a representation and record of your physical device in the cloud. Any physical device needs a thing record in order to work with AWS IoT. Learn more.

Register a single AWS IoT thing
Create a thing in your registry

Bulk register many AWS IoT things
Create things in your registry for a large number of devices already using AWS IoT, or register devices so they are ready to connect to AWS IoT.

Figure 3: Creating a Single Thing

For your first device thing, it is recommended you give it a descriptive name, such as 'CC3200_Thing' or '*MyIdName*_CC3200Board'. Leave the Thing Type, Thing Group and Attribute Key / Value options blank. Once you have entered the device name, click **Next**. When successful you should see something that looks like Figure 4.


AWS IoT
https://us-west-2.console.aws.amazon.com/iot/home?region=us-west-2#/create/single-provision
Services
Resource Groups
Lance
Oregon
Support
STEP 2/3
Create certificate
Create with CSR
Get started
Create thing without certificate
Feedback
English (US)
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

CREATE A THING

Add a certificate for your thing

A certificate is used to authenticate your device's connection to AWS IoT.

One-click certificate creation (recommended)
This will generate a certificate, public key, and private key using AWS IoT's certificate authority.

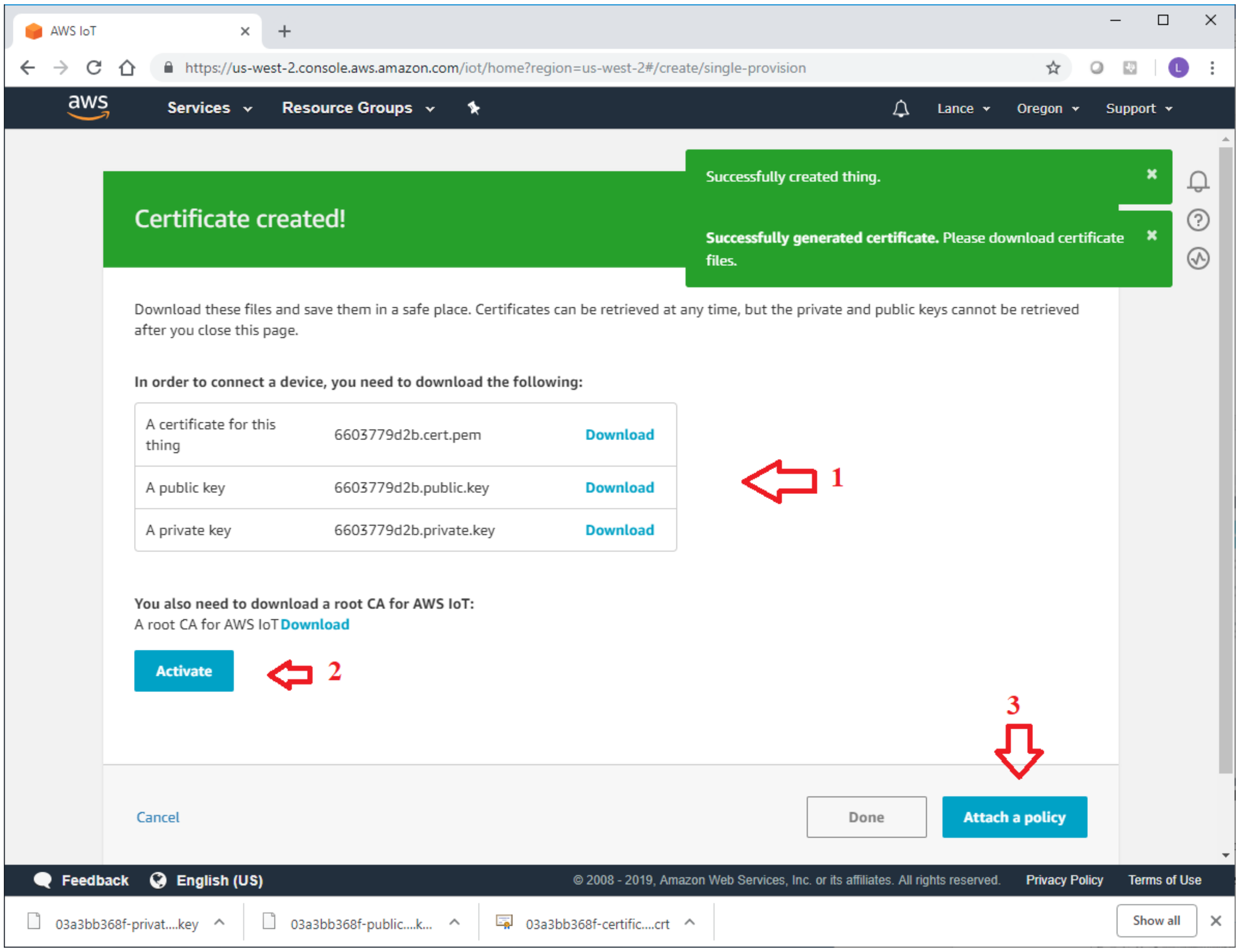
Create with CSR
Upload your own certificate signing request (CSR) based on a private key you own.

Use my certificate
Register your CA certificate and use your own certificates for one or many devices.

Skip certificate and create thing
You will need to add a certificate to your thing later before your device can connect to AWS IoT.

Figure 4: Creating Certificate for Your Thing

Click the **Create certificate** button for the One-click certificate creation as shown in Figure 4.


AWS IoT
https://us-west-2.console.aws.amazon.com/iot/home?region=us-west-2#/create/single-provision
aws
Services
Resource Groups
Lance
Oregon
Support
1
2
3
Feedback
English (US)
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
03a3bb368f-privat....key
03a3bb368f-public....k...
03a3bb368f-certific....crt
Show all

Successfully created thing.

Successfully generated certificate. Please download certificate files.

Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

A certificate for this thing	6603779d2b.cert.pem	Download
A public key	6603779d2b.public.key	Download
A private key	6603779d2b.private.key	Download

You also need to download a root CA for AWS IoT:
A root CA for AWS IoT **Download**

Activate

Cancel   Done   Attach a policy

Figure 5: Creation of Certificate, Public and Private keys

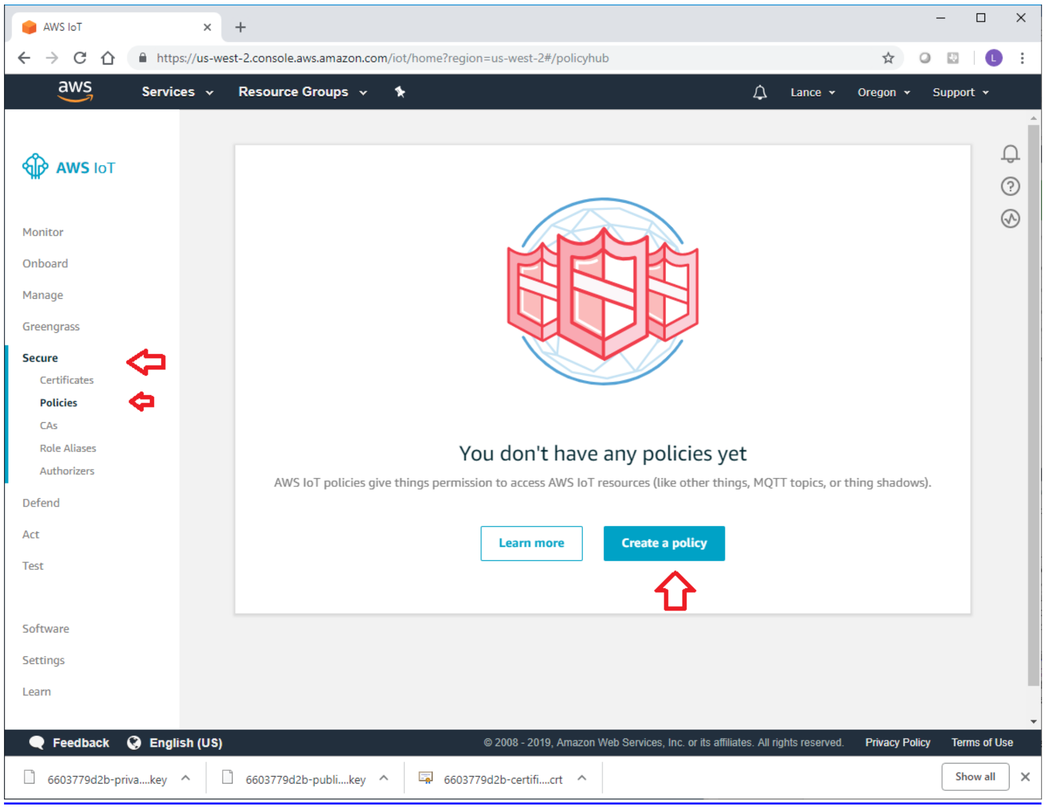
AWS should generate the required public key, private key, and certificate for identifying and encrypting traffic from your device, and allow you to download them to your computer. **It is important to download and save these files as you will NOT BE ABLE TO RETRIEVE THE PUBLIC and PRIVATE KEYS after you close the page.** We will provide you with the appropriate root CA file so don't bother downloading it from AWS. Download and save the certificate, public and private keys to a folder. Later in this tutorial, we will convert them from the .pem format to the .der format and load them into the CC3200's serial flash.

After downloading, don't forget to **Activate** the certificate. To learn more about how certificates are used, see: https://realtimelogic.com/blog/2013/10. Finally, once you have activated your certificate and keys, click the **Attach a policy** button. Since we don't have any existing policies, just click on **Register a Thing** on the next window. You have now registered your thing with an active certificate, but we still need to create and attach a policy to the certificate.

Part I.C: Making a Policy to Allow Update/Get Status from Thing Shadows

To represent your *thing* (i.e., your CC3200 board) in the cloud, AWS uses the concept of a *device shadow* to which you can get the last state or push a new state to the server. Therefore, with intermittent connection to a wireless network, your device could pull the latest state from the cloud and maintain continuity. However, appropriate security measures can be made on AWS to ensure that the devices have the *privilege* to perform the actions requested. These next steps will walk you through how to do this:

From the AWS IoT console, click on **Secure** and then **Policies**, as shown in Figure 6



https://us-west-2.console.aws.amazon.com/iot/home?region=us-west-2#/policyhub

You don't have any policies yet

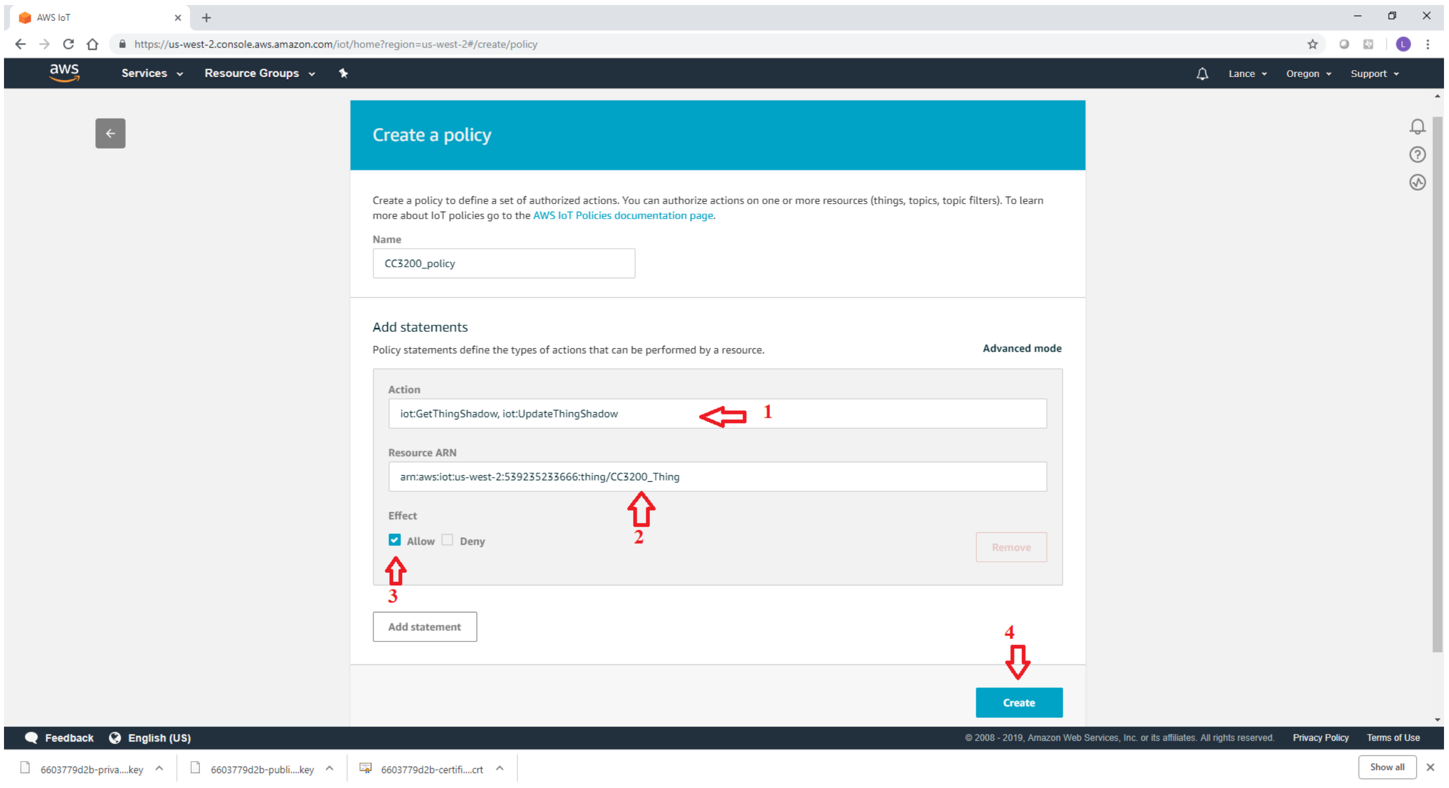
AWS IoT policies give things permission to access AWS IoT resources (like other things, MQTT topics, or thing shadows).

© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

6603779d2b-priva....key 6603779d2b-publi....key 6603779d2b-certifi....crt

Figure 6: Create a Policy

Click on **Create a policy**. This should bring up a window similar to Figure 7.



https://us-west-2.console.aws.amazon.com/iot/home?region=us-west-2#/create/policy

Create a policy

Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters). To learn more about IoT policies go to the AWS IoT Policies documentation page.

Name: CC3200_policy

Add statements

Policy statements define the types of actions that can be performed by a resource.

Action: iot:GetThingShadow, iot:UpdateThingShadow

Resource ARN: arn:aws:iot:us-west-2:539235233666:thing/CC3200_Thing

Effect: [✓ Allow] Deny

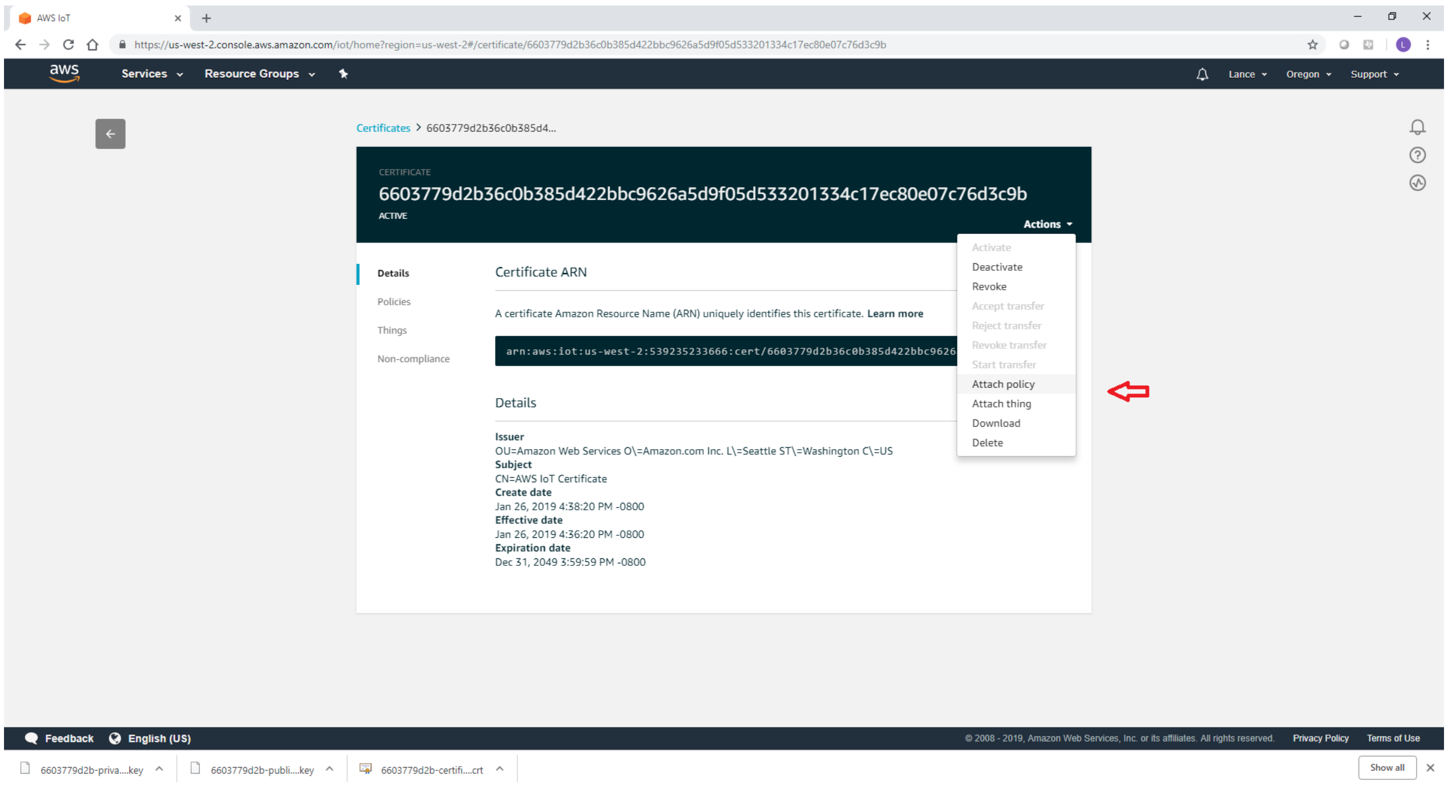
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

6603779d2b-priva....key 6603779d2b-publi....key 6603779d2b-certifi....crt

Figure 7: Creating a Policy

Give your policy a descriptive name, and add the actions "iot:GetThingShadow" and "iot:UpdateThingShadow". Furthermore, update the Amazon Resource Number (ARN) to signify that these actions apply to your *thing* created previously. Be sure to select the *Allow* Effect and then click on **Create**. You can learn more about the actions and other access management modules in the AWS documentation (http://docs.aws.amazon.com/IAM/latest/UserGuide//access_policies.html).

Now that you have created a policy, you need to attach it to your certificate. From the AWS IoT console, select **Secure** and **Certificates** and then click on you're the certificate you recently created. From the Actions pull-down menu, click on **Attach policy** as shown in Figure 8.



CERTIFICATE
6603779d2b36c0b385d422bbc9626a5d9f05d533201334c17ec80e07c76d3c9b
ACTIVE

Certificate ARN

A certificate Amazon Resource Name (ARN) uniquely identifies this certificate. Learn more

```
arn:aws:iot:us-west-2:539235233666:cert/6603779d2b36c0b385d422bbc9626
```

Details

Issuer
OU=Amazon Web Services O\=Amazon.com Inc. L\=Seattle ST\=Washington C\=US
Subject
CN=AWS IoT Certificate
Create date
Jan 26, 2019 4:38:20 PM -0800
Effective date
Jan 26, 2019 4:36:20 PM -0800
Expiration date
Dec 31, 2049 3:59:59 PM -0800

Figure 8: Attaching Policy to Certificate

Select the policy that you just created and then click on **Attach**. Now your policy should be successfully attached to the certificate for your thing. This will allow you to use the GET and POST commands via REST from the device associated with this certificate.

Part I.D: Converting the Keys/Certificates for Use with the CC3200

In this part of the lab, you will convert the private key and certificates to another format. This is necessary since AWS uses the .pem format, while your CC3200 uses the .der format.

The connection security between the CC3200 board and AWS is guaranteed through the TLS protocol. There are four components involved, the **public** and **private key**, a **certificate** and a **root certificate**. While the private key is used to decipher encrypted messages, the certificate is used to validate the public key. Certificates are built on a chain of trust, and so there is a root certificate which is used to sign the certificate.

The key and certificates can exist in either .pem or .der format. The former has ASCII-readable characters (unsecure, readable), while the latter is in binary and more compact (still unsecure, but harder to read). Your generated files in AWS are in the .pem format but the CC3200 requires the .der format.

To convert the files from .pem to .der format, we will use openSSL, an open source library for transport layer security. You need to open a command prompt and navigate to the openSSL installation directory. The commands that are needed are the following:

For the Root Certificate Authority (rootCA) file and the certificate file

```
>openssl x509 -outform der -in C:\...\rootCA.pem -out C:\...\rootCA.der
```

(Note: we will provide the rootCA.der file)

```
>openssl x509 -outform der -in C:\...\certificate.pem.crt -out C:\...\client.der
```

For the Private Key File:

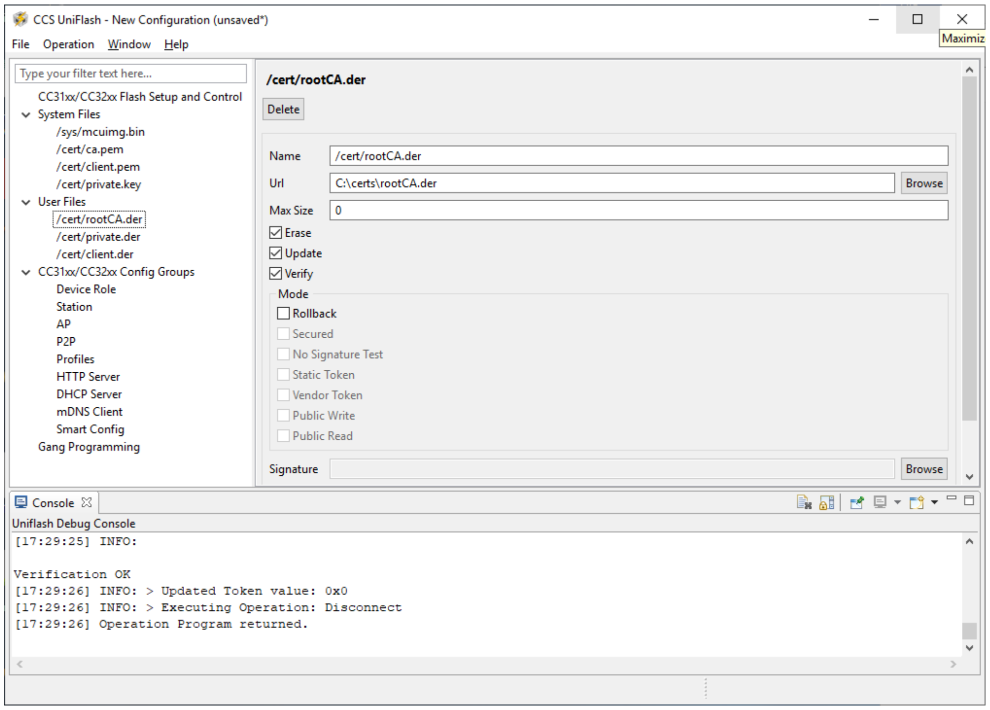
```
>openssl rsa -outform der -in C:\...\private.pem.key -out C:\...\private.der
```

You will obviously need to modify your file names and file paths to where you saved your keys/certificates. If necessary, move the files that need to be converted into the same file directory in which openSSL was installed. The rootCA file will be shared among all of the CC3200s used for the project. The client and private keys should be unique to each device. It is highly recommended to group or name the files accordingly so that the keys/certificate pairs do not get mixed up or over-written.

Part I.E. Using UniFlash to flash key and certificates to CC3200

To enable a secure connection from your CC3200, you need to download the private key and certificates to the serial flash on your board. You will do this using the UniFlash utility, as described in the Uniflash tutorial.

You should flash the .der formatted certificates, *root certificate* (used for authenticating the AWS server), *client certificate* (used by AWS to authenticate our device), and the *private key* (for device-side encryption) to **User Files** using the UniFlash utility, as shown in Figure 9. You will specify the path to those files on your PC in the Url box and select the Erase, Update, and Verify checkboxes, and click *Program* to flash those files to the board. You can also load the application (which you will develop in a later part) binary into /sys/mcuimg.bin or you can download it from Code Composer Studio and run it from the Debugger. Take note of the path of your files on the flash (i.e. – the 'Name' field below) as you will need to reference them in your code. If you forget them, you can list the files on your flash by clicking on *List Files* in UniFlash. Be sure to place jumper on SOP2 for flashing.



Name: /cert/rootCA.der
Url: C:\certs\rootCA.der
Max Size: 0
☑ Erase
☑ Update
☑ Verify
Mode
☐ Rollback
☐ Secured
☐ No Signature Test
☐ Static Token
☐ Vendor Token
☐ Public Write
☐ Public Read
Signature:

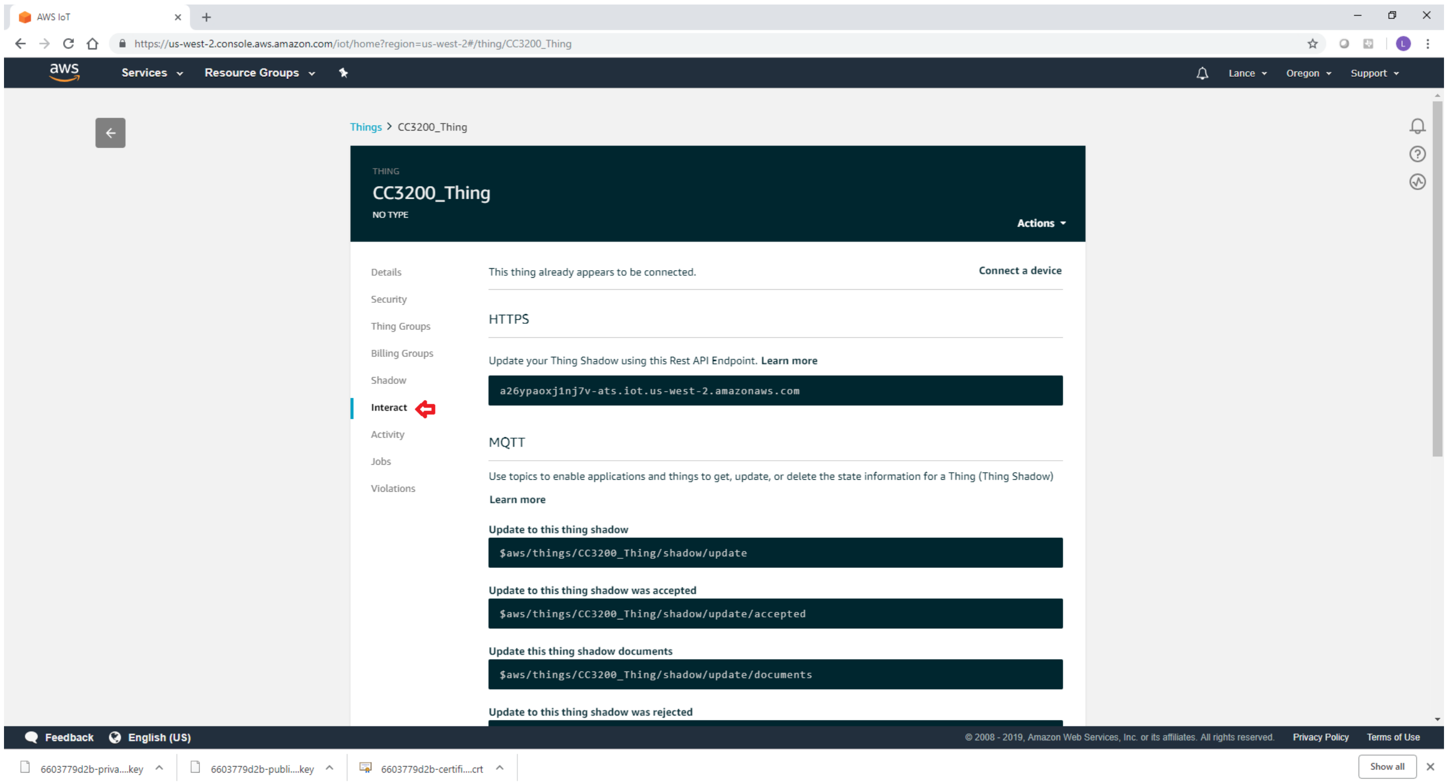
```
[17:29:25] INFO:

Verification OK
[17:29:26] INFO: > Updated Token value: 0x0
[17:29:26] INFO: > Executing Operation: Disconnect
[17:29:26] Operation Program returned.
```

Figure 9: Sample Setup to Flash a RootCA, Private Key and Certificate

Part I.F. Accessing AWS using the RESTful API

Amazon's RESTful API for AWS IoT provides access to *device shadows*, or *thing shadows*. These are JavaScript Object Notation (JSON) formatted data that allow state information for AWS things to be stored and retrieved, even when a device might be disconnected from the service. To store or retrieve data from a device shadow, Amazon provides special MQTT feeds and RESTful API calls.

Select your device thing from the AWS IoT console and click on **Interact** as shown in Figure 10.



This thing already appears to be connected.

HTTPS

Update your Thing Shadow using this Rest API Endpoint. **Learn more**

```
a26ypaoxj1nj7v-ats.iot.us-west-2.amazonaws.com
```

MQTT

Use topics to enable applications and things to get, update, or delete the state information for a Thing (Thing Shadow)

Learn more

Update to this thing shadow

```
$aws/things/CC3200_Thing/shadow/update
```

Update to this thing shadow was accepted

```
$aws/things/CC3200_Thing/shadow/update/accepted
```

Update this thing shadow documents

```
$aws/things/CC3200_Thing/shadow/update/documents
```

Update to this thing shadow was rejected

Figure 10: Rest API Endpoint for Your Thing

From this page, you can get the Rest API endpoint for your thing as well as MQTT topics that will allow you to update your thing's shadow. The REST API endpoint displays the web address for the AWS Thing/Shadow, and contains the web address of your account's AWS Endpoint. For now, copy the endpoint address into a word or text file, and save it for reference. You will need it later to connect to the AWS server. Below is an example Restful API endpoint address. The initial string of characters will be some randomly generated string generated for your account by Amazon.

Endpoint:
`identifier.iot.region.amazonaws.com`

Example:
https://xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com

Updating State Information to your Device Shadow:
`https://endpoint/things/thingName/shadow`

Example GET Method (GetThingShadow)

```
GET /things/thingName/shadow HTTP/1.1
Host: identifier.iot.region.amazonaws.com
```

```
Connection: Keep-Alive
```

Example POST Method (UpdateThingShadow)

```
POST /things/thingName/shadow HTTP/1.1
Host: identifier.iot.region.amazonaws.com
Connection: Keep-Alive
Content-Type: application/json; charset=utf-8
Content-Length: 30
{
      "state":{
            "desired":{
                  "color":"green"
            }
      }
}
```

To send a RESTful API POST message to AWS, the service must first be connected to with the SimpleLink socket library. The initial connection should be to the base address for the service or website with which you are trying to communicate. For example, you may have an AWS device thing that is said to have the RESTful API address of the following: https://<endpoint>.iot. us-west-2.amazonaws.com/things/CC3200_Thing/shadow. The initial connection with the socket library should only be to <endpoint>. iot. us-west-2.amazonaws.com. The HTTPS part of the address will be managed based on the socket credentials, and the AWS device thing subdirectory will be used in the RESTful API calls.

Connection Configuration for AWS

- TLS1.2
- Cipher: TLS ECDHE RSA with AES 256 CBC SHA
- Port: 8443
- Certificates needed: Certificate Authority file for AWS (rootCA.der), x509 Device Certificate file (client.der) and the matching private key file (private.der)

The RESTful API application requires Internet connectivity through an Access Point (AP). The AP details are contained in the header file common.h. Modify the common.h header file, if necessary, to match the following configuration for the AP in the lab:

```
// Values for below macros shall be modified as per access-point(AP) properties
// SimpleLink device will connect to following AP when application is executed
//
#define SSID_NAME           "eec172"          /* AP SSID */
#define SECURITY_TYPE       SL_SEC_TYPE_OPEN  /* Security type (OPEN or WEP or WPA*/
#define SECURITY_KEY        ""                /* Password of the secured AP */
#define SSID_LEN_MAX        32
#define BSSID_LEN_MAX       6
```

Once common.h has been modified, you can build and test the example program. Please note that common.h does not reside in your workspace, so if you work on the lab computers *and* your personal laptop you will need to modify them in both places, and with the appropriate settings for the wireless network you are trying to connect with.

Another important change is that the secure client application must have the "current" date and time so that the credentials can be verified as valid. In this example, we use variables in the main program to update the date and time. The date and time only need to be current within the last month or so. Having the date and time hard-coded into the program is obviously not a robust or secure programming method since the code will eventually stop working unless periodically updated. However, in the interest of reducing complexity, we will use this method for this lab. The date and time variables are stored in the following constants in main.c:

```
//NEED TO UPDATE THIS FOR IT TO WORK!
#define DATE                26   /* Current Date */
#define MONTH               01     /* Month 1-12 */
#define YEAR                2019  /* Current year */
#define HOUR                17    /* Time - hours */
#define MINUTE              0    /* Time - minutes */
#define SECOND              0     /* Time - seconds */
```

Now you can connect your CC3200 securely to AWS. Demonstrate that you can use a POST request to post state to your AWS thing and use a GET request to obtain status information on your AWS thing. You should receive an HTTP 200 status code, showing a successful POST or GET request, as well as some basic status information about your device thing.

NOTE: you need to give your thing shadow some initial state before you can do an HTTP GET on it. You can give the shadow state by using the HTTP POST command or by using the AWS web interface. Also, update the time in main.c everytime you run your program. The time difference of few minutes should be ok. In common.h file, SSID_Name is the name of your WiFi. We suggest that you use portable WiFi hotspot on your device with open security. The CC3200 doesn't require a lot of data to work with AWS. If this is not possible, then change SECURITY_TYPE to match to your WiFi and enter the WiFi password in SECURITY_KEY.

In this part, you can start with the working example project **SSL_REST_API_AWS_W19.zip**. You will need to examine it carefully to understand it and modify it appropriately. In particular, you should examine the http_post() function to see how the HTTP command is sent to AWS. The data must be formatted properly, or AWS (or any other web service) will not accept it as valid. In order to use this project on your system, you may need to modify the following files or parameters:

1. common.h – as described above for the eec172 Access Point.
2. main.c – you will need to use your own AWS endpoint path and thing name as well as changing the HTTP command in order to test both GET and POST. You should also change the message data that you post to your thing. However, be careful to keep the correct data format. You may also need to update the date, as described above.
3. pinmux.c – verify that the pinmux.c file sets up the pins that you intend to use. For the default application, only the UART0 and GPIO signals driving the on-board LED are actually used. If your program uses the same pins as the on-board LEDs, you will have problems.
4. Flash the private key, the client certificate and the root Certificate Authority into the CC3200 flash, as described.

Part II: Using SNS to send a text to your phone

For this project you are to compose a text message using your IR remote and send it to your phone. This will be accomplished by setting up a topic in the Simple Notification Service and registering your text-capable cell phone, and establishing a Rule in the IoT service to trigger when you update your device shadow and send messages to your phone. The interactive tutorial you went through in **Part I.A** gives you a high-level overview on how this is accomplished in AWS.

For this lab we recommend that you use the "US West (Oregon)" AWS server. You need to use a server that runs both the IoT and SNS services.

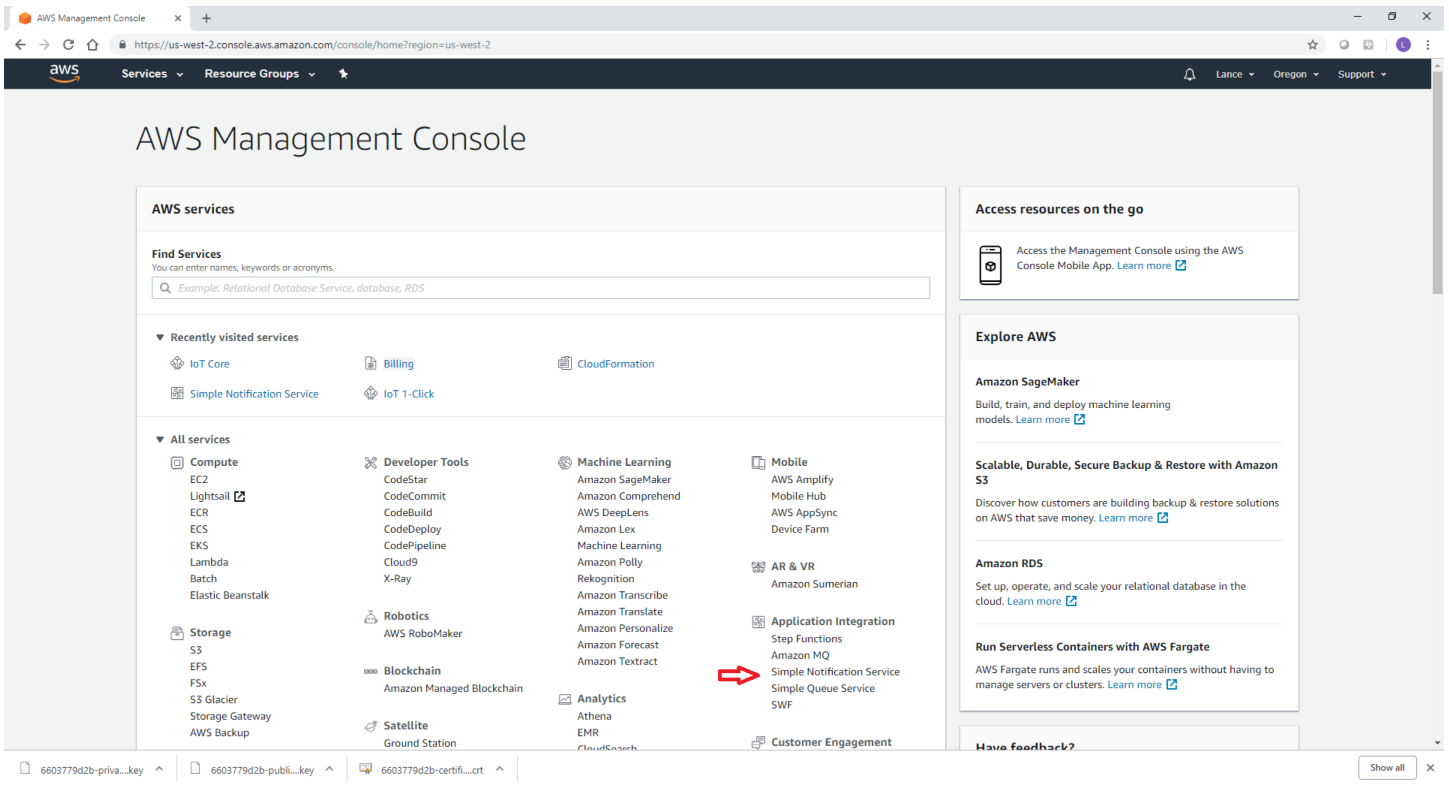
To do this you must perform the following:

1. Create a thing for the cc3200 as you have done before.
2. Create an SNS topic
3. Subscribe to the SNS topic with your phone number and the SMS option.
4. Create a rule to forward incoming thing messages to your SNS topic.
5. Use the REST API to post to your topic.

We will walk you through **creating an SNS topic** and **creating a rule.**

Part II.A. Creating an SNS Topic:

Amazon Web Services has a collection of different web services which can be integrated to perform a larger function. So far you have been using the IoT service to communicate the states of *real-world devices or things* to the cloud. The Simple Notification Service (SNS) is another module that will allow you to push messages to subscribers. Navigate to the SNS service by expanding the services tab and finding Simple Notification System under *Application Integration* as shown in Figure 11.



Access the Management Console using the AWS Console Mobile App. Learn more

Build, train, and deploy machine learning models. Learn more

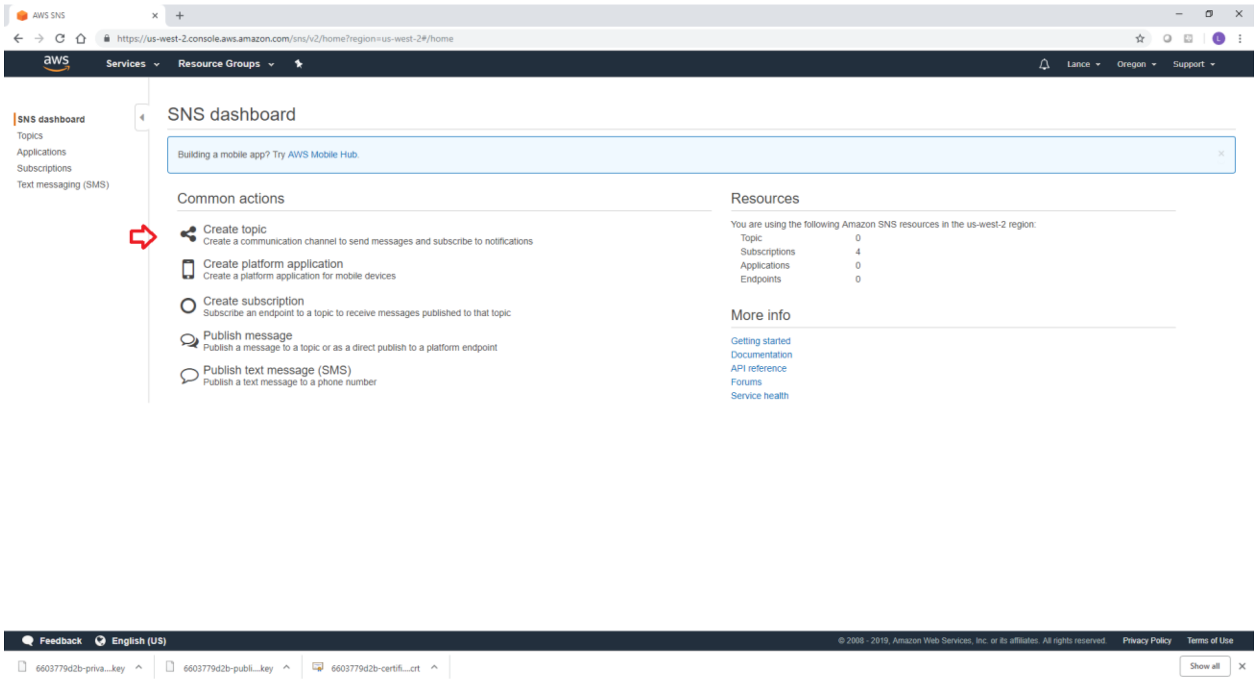
Discover how customers are building backup & restore solutions on AWS that save money. Learn more

Set up, operate, and scale your relational database in the cloud. Learn more

AWS Fargate runs and scales your containers without having to manage servers or clusters. Learn more

Figure 11: Simple Notification Service in Menu of AWS Services

From the SNS dashboard, click on **Create topic** as shown in Figure 12.



SNS dashboard

Building a mobile app? Try AWS Mobile Hub.

Create topic
Create a communication channel to send messages and subscribe to notifications

Create platform application
Create a platform application for mobile devices

Create subscription
Subscribe an endpoint to a topic to receive messages published to that topic

Publish message
Publish a message to a topic or as a direct publish to a platform endpoint

Publish text message (SMS)
Publish a text message to a phone number

You are using the following Amazon SNS resources in the us-west-2 region:

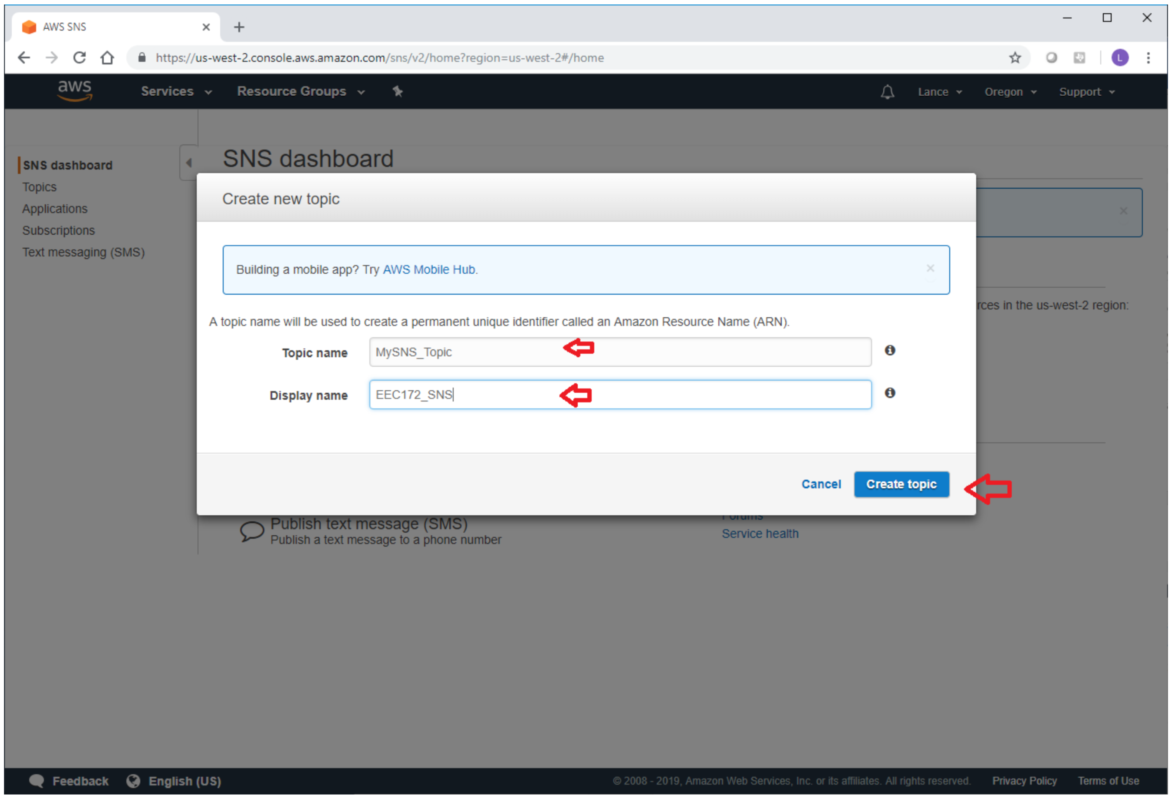
Topic	0
Subscriptions	4
Applications	0
Endpoints	0

© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

6603779d2b-priva....key 6603779d2b-publi....key 6603779d2b-certifi....crt

Figure 12: SNS Dashboard

Next, choose a Topic name and a Display name of no more than 10 characters and then click **Create topic**, as shown in Figure 13.



Create new topic

Building a mobile app? Try AWS Mobile Hub.

A topic name will be used to create a permanent unique identifier called an Amazon Resource Name (ARN).

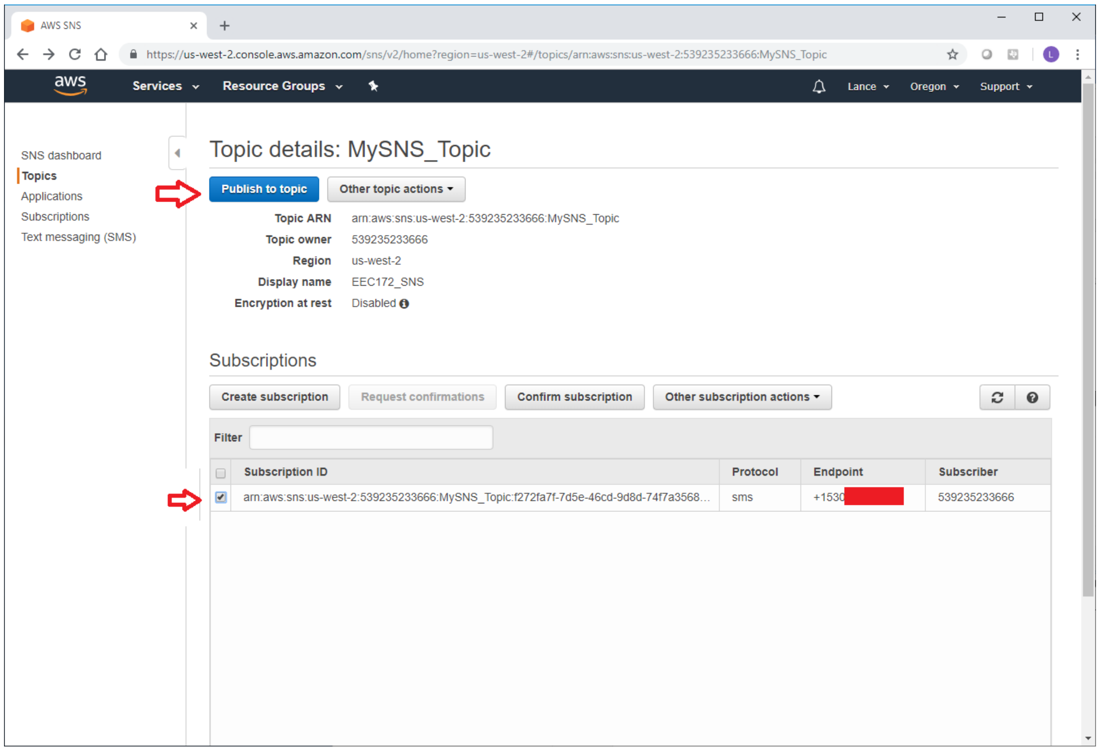
Topic name: MySNS_Topic

Display name: EEC172_SNS

© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 13: Creating an SNS Topic

Afterwards, **Create a Subscription**, where the protocol is *SMS* (Simple *Messaging* Service) for text messages. The endpoint is your cell-phone number to receive text messages. You can then **test** to make sure this module works by clicking on the '**Publish to topic**' button and manually sending a text message from the AWS Console to your cell-phone (use RAW format for now).



Topic details: MySNS_Topic

Topic ARN	arn:aws:sns:us-west-2:539235233666:MySNS_Topic
Topic owner	539235233666
Region	us-west-2
Display name	EEC172_SNS
Encryption at rest	Disabled

Subscriptions

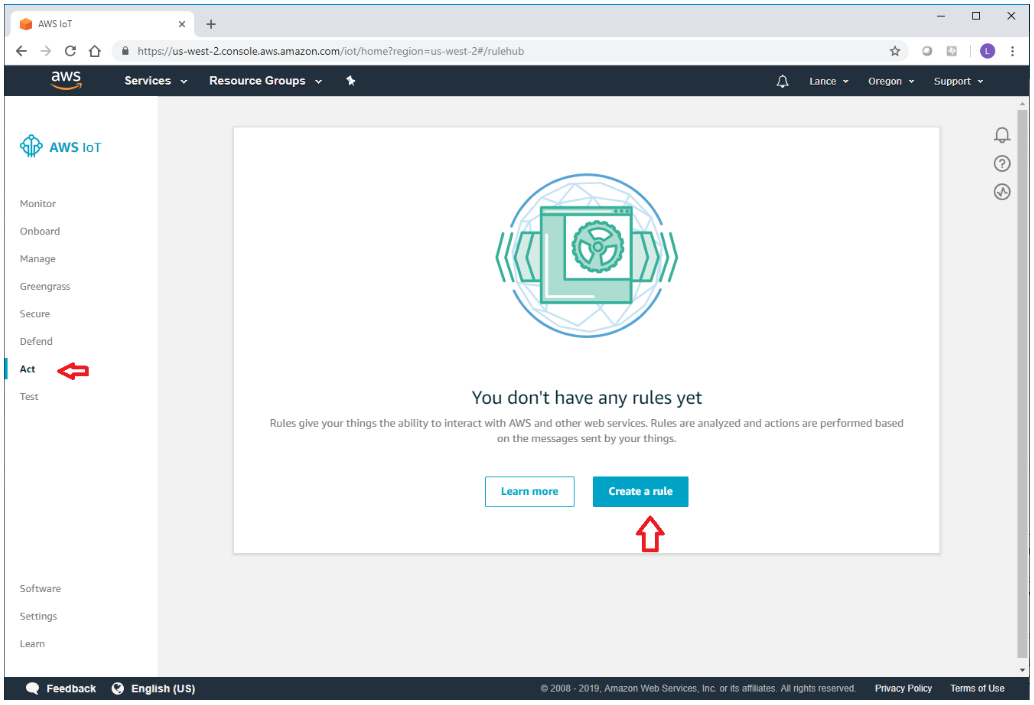
	Subscription ID	Protocol	Endpoint	Subscriber
✓	arn:aws:sns:us-west-2:539235233666:MySNS_Topic:f272fa7f-7d5e-46cd-9d8d-74f7a3568…	sms	+1530[redacted]	539235233666

Figure 14: Publish a Message to your SNS Topic

Part II.B. Creating an IoT Rule:

The next step is to create an IoT Rule that will trigger when you push updates to your shadow device. Earlier in the lab, we updated our device shadow (state) via the REST API. When our device shadow is updated, the IoT module publishes messages to several reserved Message Queue Telemetry Transport (MQTT) topics, a publish-subscribe 'lightweight' messaging protocol. In order to trigger your rule, we will be listening for updates to the device shadow via these topics. A full list of reserved topics that can be subscribed to can be found at http://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-mqtt.html .

From the AWS IoT console, click on **Act** and then click on **Create a Rule**, as shown in Figure 15.

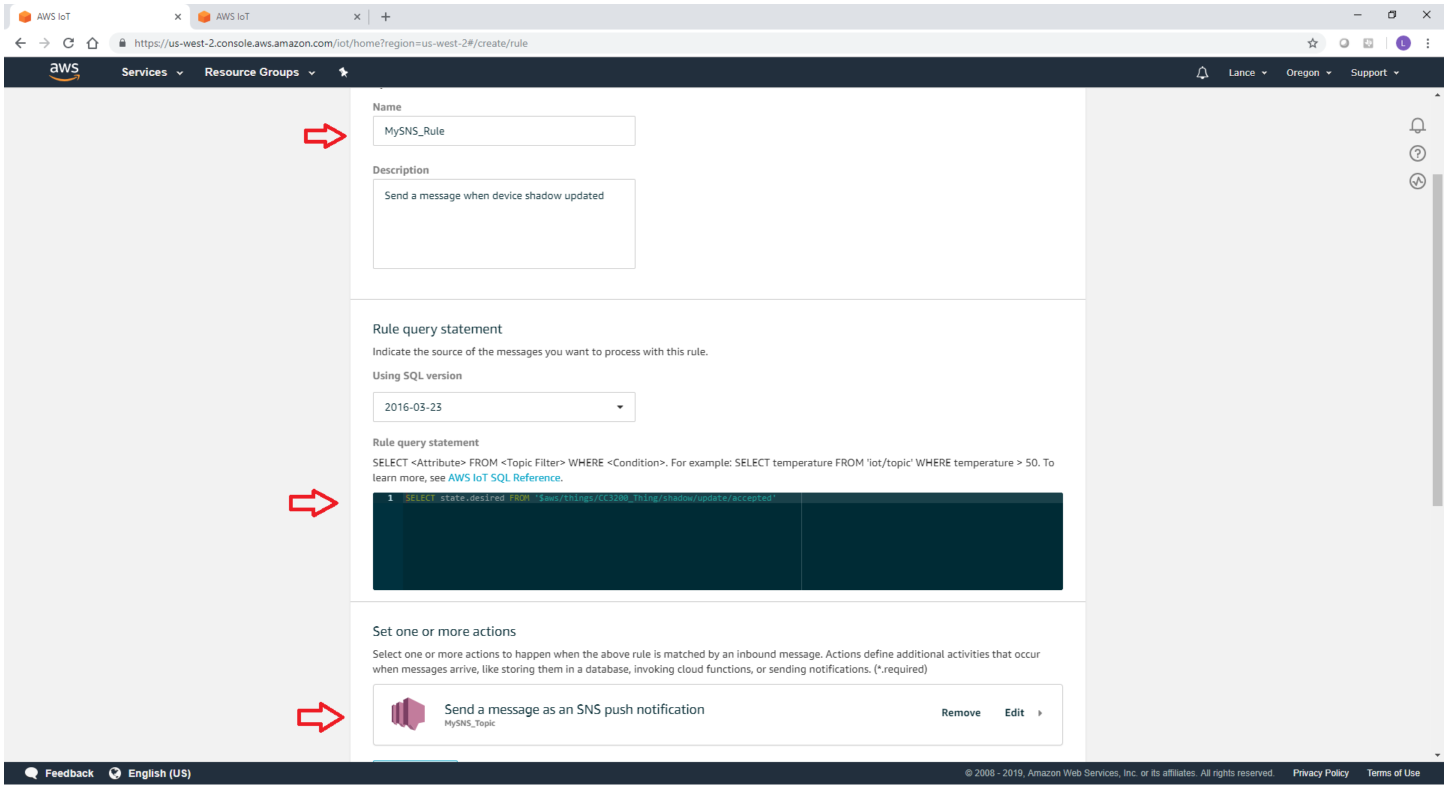


You don't have any rules yet

Rules give your things the ability to interact with AWS and other web services. Rules are analyzed and actions are performed based on the messages sent by your things.

Figure 15: Creating a Rule

You should listen to the topic '`$aws/things/`*`thingName`*`/shadow/update/accepted`' (the $ and the single quotes should be *included*), report on the attribute '`state.desired`', and leave the condition blank. Then attach an action that will send an SNS push notification when the rule is triggered. Select the SNS target to be the topic you created previously, and use the 'RAW' message format (for now). You will also need to create an IAM role to allow the IoT to access the SNS service 'securely' (you can learn more about Identity Access and Management [IAM] roles here: http://docs.aws.amazon.com/iot/latest/developerguide/iam-users-groups-roles.html, but will not need to understand them for this lab). Your SQL query statement for your rule should be similar to the figure below.



Name: MySNS_Rule

Description: Send a message when device shadow updated

Rule query statement

Indicate the source of the messages you want to process with this rule.

Using SQL version: 2016-03-23

Rule query statement

SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see AWS IoT SQL Reference.

```
SELECT state.desired FROM '$aws/things/CC3200_Thing/shadow/update/accepted'
```

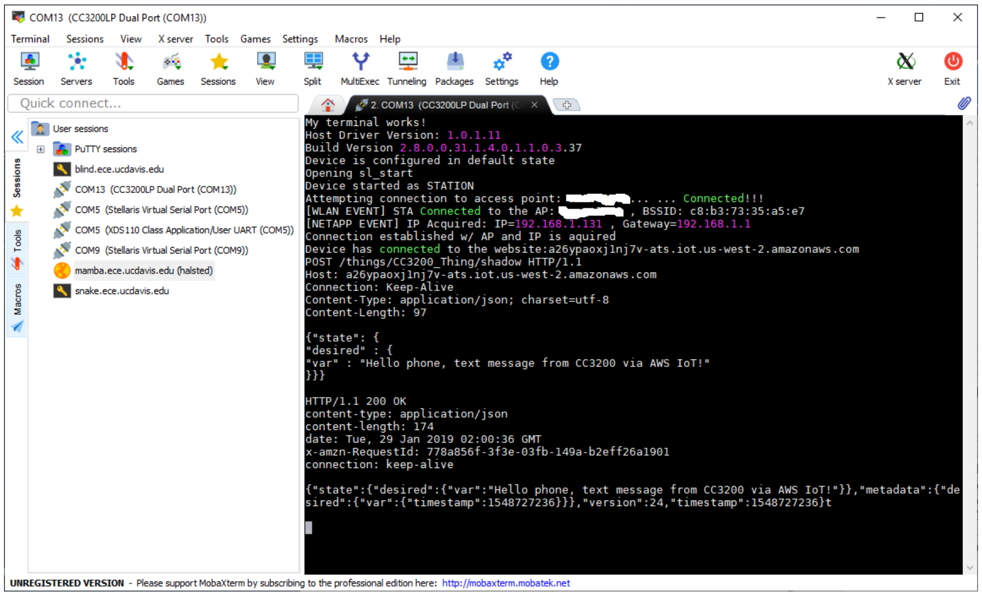
Set one or more actions

Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. (*.required)

Send a message as an SNS push notification — MySNS_Topic

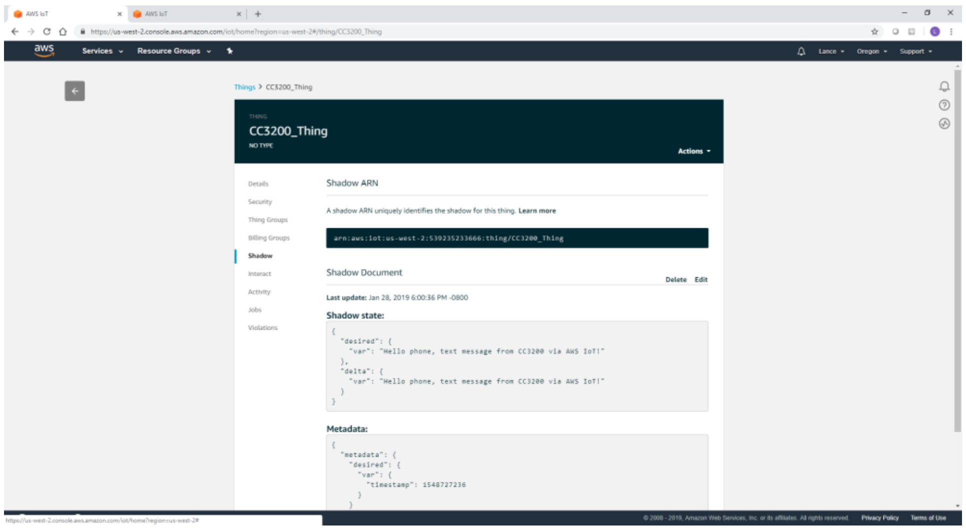
Figure 16: Setting up your IoT Rule

Make sure the rule is enabled and try to push an update to your device shadow using your CC3200,… in a couple of seconds you *should* get a text message. Once that works, you can clean up the formatting using JSON message format SNS action. To figure out how to do this, go back to the SNS topic you made, click the *Publish a topic* button, and experiment with the JSON message generator to find the appropriate syntax. Investigate how you will need to modify your SQL statement to 'clean-up' your message. Helpful link and subsections for SQL: http://docs.aws.amazon.com/iot/latest/developerguide/iot-sql-reference.html


COM13 (CC3200LP Dual Port (COM13))
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick connect...
User sessions
PuTTY sessions
blind.ece.ucdavis.edu
COM13 (CC3200LP Dual Port (COM13))
COM5 (Stellaris Virtual Serial Port (COM5))
COM5 (XDS110 Class Application/User UART (COM5))
COM9 (Stellaris Virtual Serial Port (COM9))
mamba.ece.ucdavis.edu (halsted)
snake.ece.ucdavis.edu
2. COM13 (CC3200LP Dual Port (C
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

```
My terminal works!
Host Driver Version: 1.0.1.11
Build Version 2.8.0.0.31.1.4.0.1.1.0.3.37
Device is configured in default state
Opening sl_start
Device started as STATION
Attempting connection to access point: [redacted]... ... Connected!!!
[WLAN EVENT] STA Connected to the AP: [redacted] , BSSID: c8:b3:73:35:a5:e7
[NETAPP EVENT] IP Acquired: IP=192.168.1.131 , Gateway=192.168.1.1
Connection established w/ AP and IP is aquired
Device has connected to the website:a26ypaoxj1nj7v-ats.iot.us-west-2.amazonaws.com
POST /things/CC3200_Thing/shadow HTTP/1.1
Host: a26ypaoxj1nj7v-ats.iot.us-west-2.amazonaws.com
Connection: Keep-Alive
Content-Type: application/json; charset=utf-8
Content-Length: 97

{"state": {
"desired" : {
"var" : "Hello phone, text message from CC3200 via AWS IoT!"
}}}

HTTP/1.1 200 OK
content-type: application/json
content-length: 174
date: Tue, 29 Jan 2019 02:00:36 GMT
x-amzn-RequestId: 778a856f-3f3e-03fb-149a-b2eff26a1901
connection: keep-alive

{"state":{"desired":{"var":"Hello phone, text message from CC3200 via AWS IoT!"}},"metadata":{"de
sired":{"var":{"timestamp":1548727236}}},"version":24,"timestamp":1548727236}t
```

Figure 17: Updating Device Shadow from CC3200



CC3200_Thing

Shadow ARN

A shadow ARN uniquely identifies the shadow for this thing. **Learn more**

```
arn:aws:iot:us-west-2:539235233666:thing/CC3200_Thing
```

Shadow Document

Last update: Jan 28, 2019 6:00:36 PM -0800

Shadow state:

```
{
  "desired": {
    "var": "Hello phone, text message from CC3200 via AWS IoT!"
  },
  "delta": {
    "var": "Hello phone, text message from CC3200 via AWS IoT!"
  }
}
```

Metadata:

```
{
  "metadata": {
    "desired": {
      "var": {
        "timestamp": 1548727236
      }
    }
```

Figure 18: Verifying Device Shadow State

Part II.C. Integrate your game to send a message to your phone:

Now that you have the SNS module and IoT Rules set up appropriately, you can start integrating your Lab 3 code to send a text to your phone when the game is finished. Start with the program for connecting to AWS via the REST API that you implemented in the last part. When you are done, you will be sending a text from your CC3200 through the cloud to your cellphone. Cool, yeah?

The Lab 3 has the game, which ends when the ball touches the boundary of the OLED Screen. At this point you can simply text Game Ended! to your phone.

Note: If the SMS is not getting delivered due to some issues, you can also use email. This step has to be done while **Create a Subscription.**

The following links provide useful information on AWS.

Amazon IoT Rules Documentation: http://docs.aws.amazon.com/iot/latest/developerguide/iot-rules.html
Amazon Device Shadows Documentation: http://docs.aws.amazon.com/iot/latest/developerguide/iot-thing-shadows.html

Demonstrate your working program to your TA and have him sign your verification sheet.

Lab Report

1. The video in *.mp4 format showcasing the Program that sends the message to the Cell phone/Email using AWS service. The video should also capture the messages printed via UART on the terminal. Please summarize your understanding of the **all AWS modules** we have used, and how the program is working.

2. The video should also capture you while explaining and demonstrating the working of the circuit. If you have made extra connections, please explain it to us through the video. For example, you can ask someone at home to film you or use the selfie camera to record yourself explaining the working and then use the back camera to capture the demonstration of the circuit. You can also create your own PowerPoint slides and walk us through it for explaining the circuit connection and your idea.

3. Try to limit the video to maximum of 10 minutes. The video and audio clarity should be decent enough for us to judge your work.

4. Zip the source files and video and upload it to the canvas, make sure the zip is not locked and the source files can be opened using text editor software. Please comment the code wherever necessary.

References and Links:

CC32xx SSL Demo Application:
http://processors.wiki.ti.com/index.php/CC32xx_SSL_Demo_Application

Configuring the AWS CLI client to connect to the Amazon Servers:
http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html

Tutorial for using the Identity and Access Management Guide (This is used to generate the credentials to connect via AWS CLI):
http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

IAM tutorial link:
http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

Amazon IoT Quickstart Guide Link:
https://docs.aws.amazon.com/iot/latest/developerguide/iot-quickstart.html
http://docs.aws.amazon.com/iot/latest/developerguide/protocols.html